Lawrence Livermore National Laboratory

How many ways can you slice a classifier?

Exploring HPC architectures and programming models for data analytics

Maya B. Gokhale

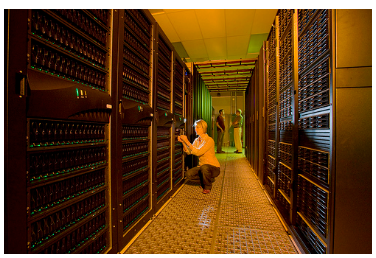
Lawrence Livermore National Laboratory

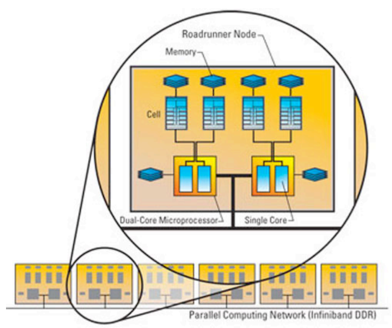
This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

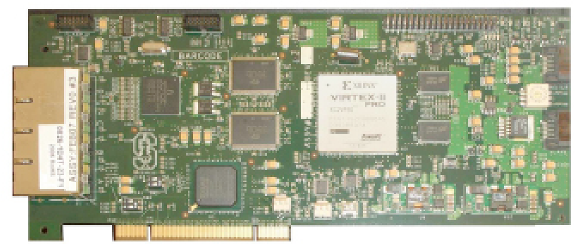
HPC architectures: simulation vs. analytics

TLCC SU compute node
- dual socket, quad core Xeon
- 8GB RAM
- 4x DDR IB (peak 16Gb/s)
- no local storage
- I/O node connects to 47GB/s Lustre storage

Yahoo terabyte sort cluster node
- dual socket, quad core Xeon
- 8GB RAM
- 1 Gb Ethernet
- 4 SATA disks/node







HPC Programming models: simulation vs. analytics

HPC simulations

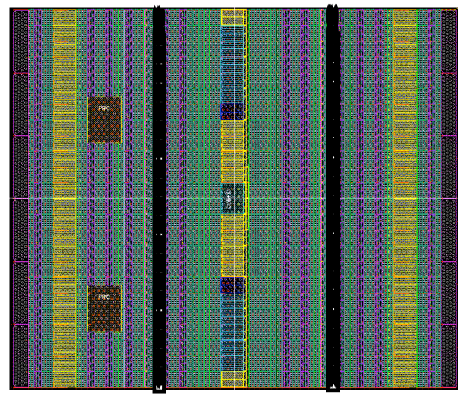
- primarily SPMD programming model supported by MPI
- state is held in memory across all the nodes
- nodes participate in periodic message exchange
- I/O to load parameters and to write checkpoint files
- favored by DOE community

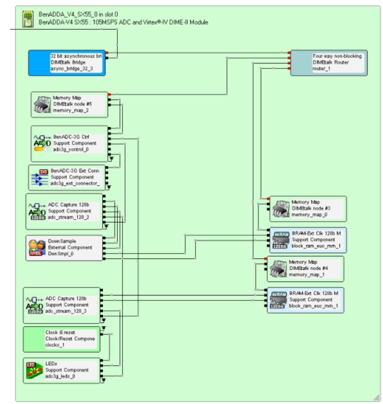
HPC analytics

- SPMD for analysis with Map/Reduce
- streaming for data ingest, processing
  - tightly coupled pipelines and data flow graphs
- I/O is integral to computation
- widespread use of commercial databases and business intelligence products

Hardware assist for streaming analytics

- FPGA
  - hardware captures signal, network packet
  - analytics pipeline is customized to the application
  - many configurations
    - PCI-E, GigE, A/D





Tilera

8 x 8 custom processors
local cache, shared memory
mesh interconnection network
many configurations
PCI-E, GigE

Case study: background

- Cybersecurity research
  - advanced analytic processing of streaming data
  - forensic analysis of pcap files
- Classifier to detect malicious HTTP get requests
- Algorithm: **Brian Gallagher, Tina Eliassi-Rad**
- Hadoop: **Tamara Dahlgren**
- Tilera: **Phil Top**
- FPGA: **Craig Ulmer** (Sandia)

Malicious HTTP request classifier

- HTTP is the universal conduit for web traffic
  - Simple, plain-text formatting
  - Gateway to databases, files, executables
- Malicious users also use these interfaces
  - Query a DB, invoke commands
  - Obfuscate commands, game network filters
- Can we detect attacks forensically?
- Can we detect attacks on the wire?

ECML/PKDD 2007 Discovery Challenge

- HTTP Traffic Classification
  - Apply machine learning to identify malicious activity in HTTP
- Hand-labeled datasets of HTTP flows
  - Training: 50K inputs, 30% attacks
  - Competition: 70K inputs, 40% attacks
  - 7 Attack Types XSS, SQL/LDAP/XPATH injection, path traversal, command execution, and SSI

Flow Example

```
GET /eH/first_str/2hFnull6/oixsotcwrseamgit2/38PrR_Lkmmzo.htm
Host: www.a215Een.st:15
Connection: close
Accept: */*
Accept-Charset: *;q=0.4
Accept-Encoding: *
Accept-Language: boHEor-sen0, gte-ltmse4      oS, 3TeoUsHn-asrao;q=0.2, paly-wreihi, 78iiqths-ar;q=0.3
Cache-Control: no-store
Client-ip: 200.91.18.159
Cookie: uciy2kleicl=%3C%21--+%23odbc+++++++++++++connect%3D%226at8h%2CHcteil%2CeHnNa%22+++++statement%3D%22drop+table+elkbO…
```

odbc

connect

statement

drop table

Gallagher/Eliassi-Rad approach

- All HTTP requests of a particular attack type constitute a single document
- In training phase, compute a TF/IDF vector for all the terms of each attack “document”
- On the testing data set of HTTP requests, compute the TF/IDF of each request “document”
- Classify the test data HTTP request according to the closest match to attack TFIDFs

TF/IDF

- Well-know information retrieval metric
- Term-Frequency, Inverse Document Frequency
  - TF: How often does each term appear in a document?
  - IDF: How specific is the term to the document?
- Cosine Similarity
  - Vector dot product to estimate angle between input and attack

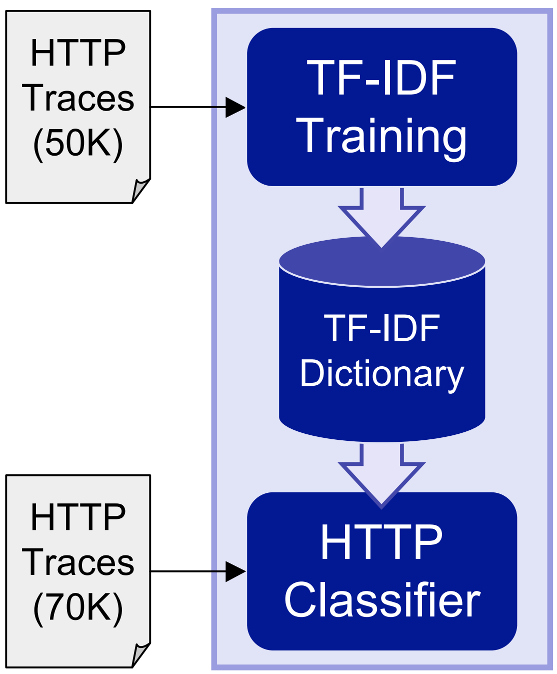
Salton, Gerard and Buckley, C. (1988). "Term-weighting approaches in automatic text retrieval". *Information Processing & Management* , 24 (5): 513–523.

$$tfidf(t,d) = \underbrace{\frac{count(t,d)}{\sum_{v \in d} count(v,d)}}_{Term\ Frequency} \cdot \underbrace{\log \frac{|D|}{|\{d_j : t \in d_j\}|}}_{Inverse\ Document\ Frequency}$$

$$\mathrm{sim}_{\cos}(a,R) = \frac{\vec{a} \cdot \vec{R}}{\|\vec{a}\| \cdot \|\vec{R}\|} = \frac{\sum_{t \in a \cap R} tfidf(t,a) \cdot tfidf(t,R)}{\sqrt{\sum_{t \in a} tfidf(t,a)^2} \cdot \sqrt{\sum_{t \in R} tfidf(t,R)^2}}$$

LLNL Approach Achieved 95% Accuracy

- Brian Gallagher and Tina Eliassi-Rad LLNL-PRES-408823
- Vector approach
  - Tokenize input
  - Assign weights to tokens via TF-IDF
  - Cosine similarity for vector comparison
- Relies on a data dictionary
  - Generate term statistics during training
  - Reference statistics at runtime



Top 3 SSI Classifier Terms

Term	IDF	Weight
odbc	2.079	0.0134
statement	2.079	0.0134
--	0.988	0.0126

Top 3 OS Commanding Classifier Terms

Term	IDF	Weight
..	1.386	0.0057
dir	2.079	0.0053
/c	2.079	0.0051

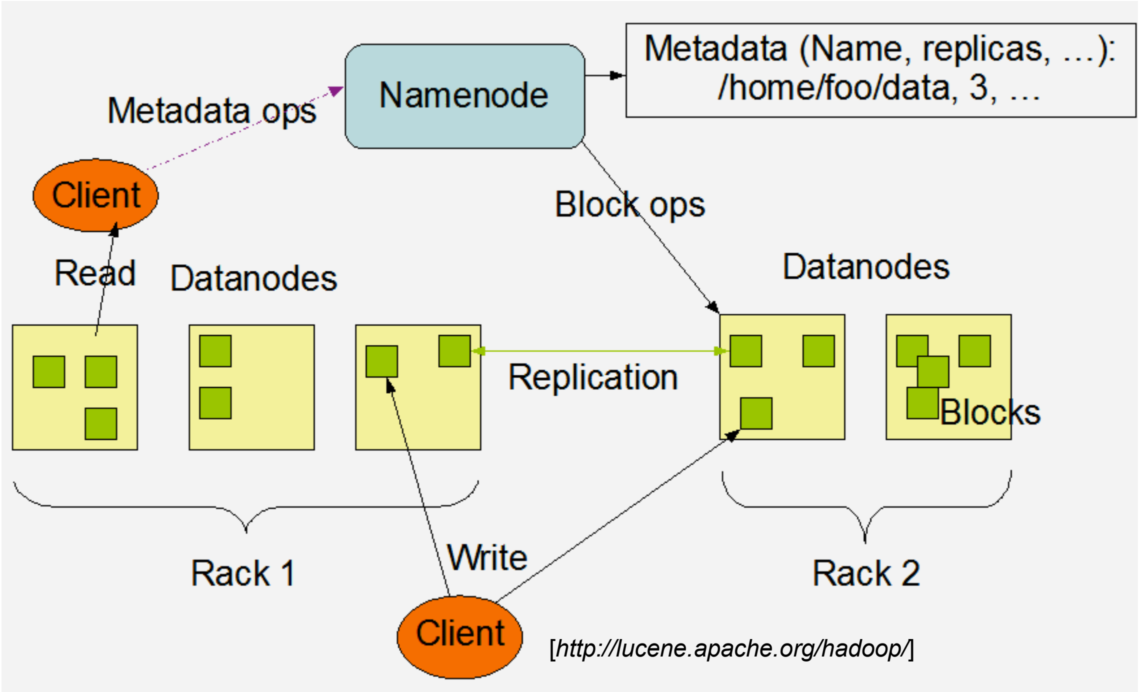
Data intensive parallel architectures for TFIDF

- Hadoop cluster
  - data parallel programming environment with structured compute-scatter/gather phases
  - suitable for retrospective analysis
- Tilera chip
  - 64-core chip derived from MIT RAW architecture supporting linux/C environment
  - supports streaming computation, particularly for network packets
- FPGA
  - versatile programmable logic chip
  - supports a variety of data flow patterns, especially streaming
  - complex tool chain - hardware is ultimately generated

Hadoop Distributed File System (HDFS)

Design Emphasis:

- Centralized Namenode for metadata operations
- Fault tolerance: data redundancy
- Write once, Read many for large files split across Data Nodes
- "Moving Computation is Cheaper than Moving Data"



TFIDF on Hadoop cluster

Java implementation

• wrapped in map/reduce framework

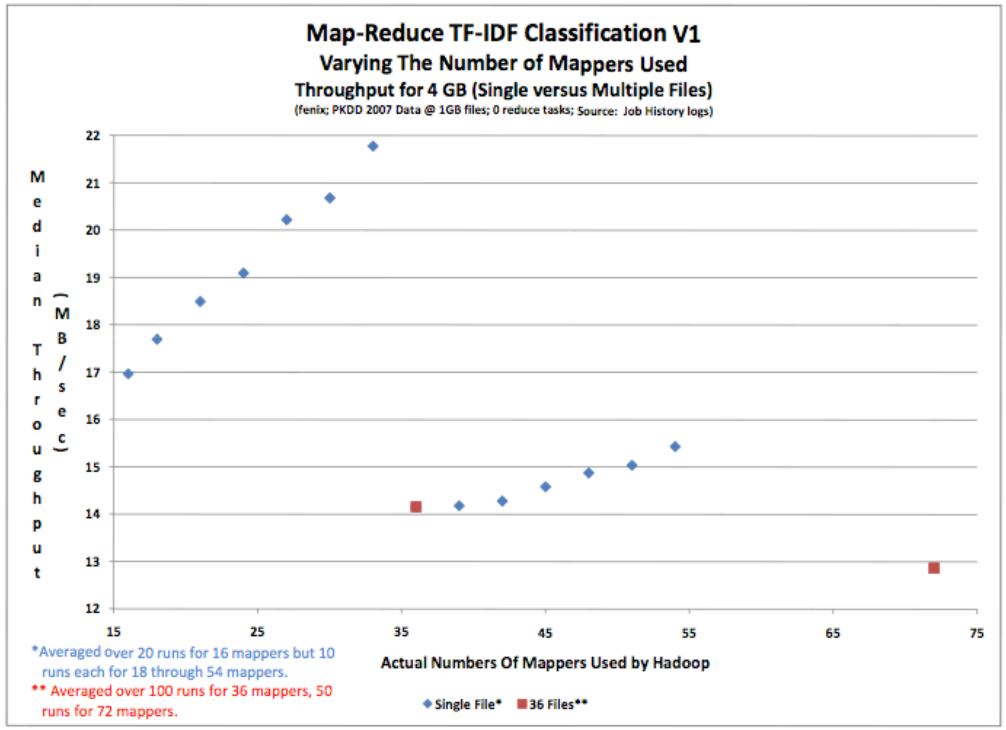
• each mapper processes an input split

• 19 worker nodes, 1 namenode

•Two Intel Xeon 2.40GHz CPUs,4GB RAM and 1 local hard disk at 80GB

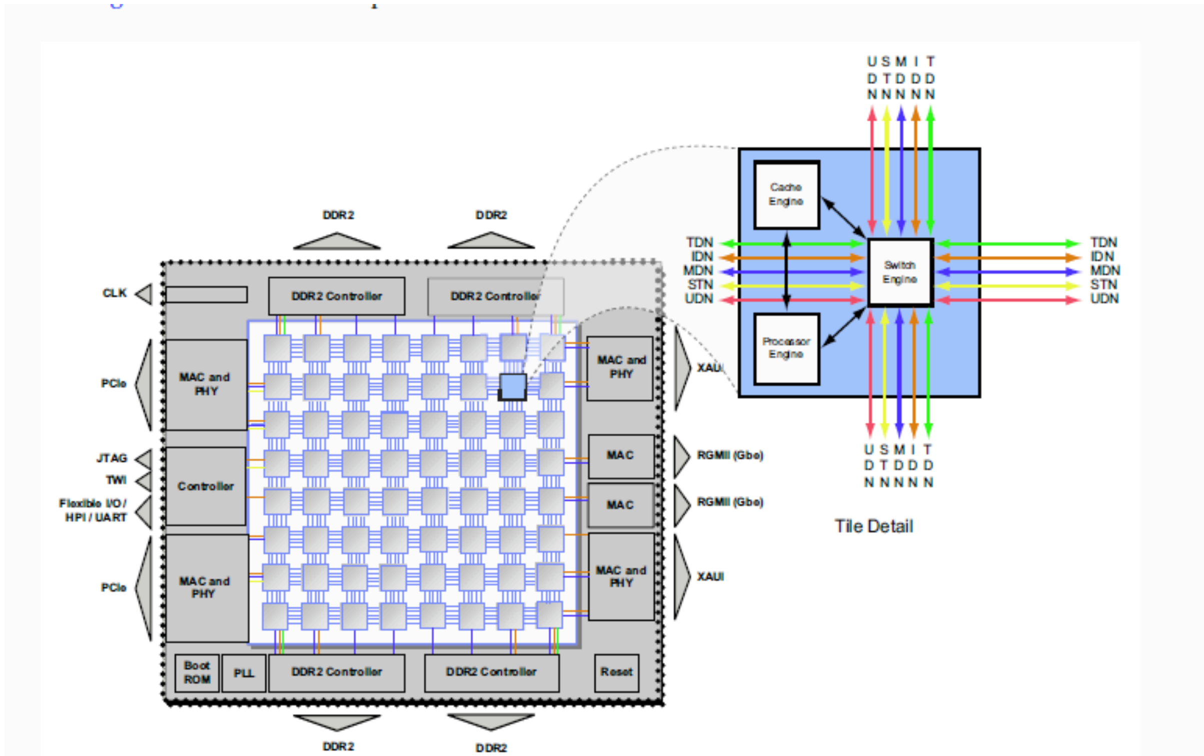
• original Java program runs at ~1MB/s.

•**Tammy Dahlgren, LLNL**



Tilera

- 8x8 array of 700 MHz custom 32-bit integer processors, runs Linux
- Custom 2D on-chip switched mesh interconnect with 5 communication networks
  - 4 dynamic, 1 static user controlled communication
  - Memory, cache operations
  - IO operations



- Chip includes 10 Gb ethernet port, PCI express ports, DDR2 memory controller
- Card has 6 1Gb ethernet ports

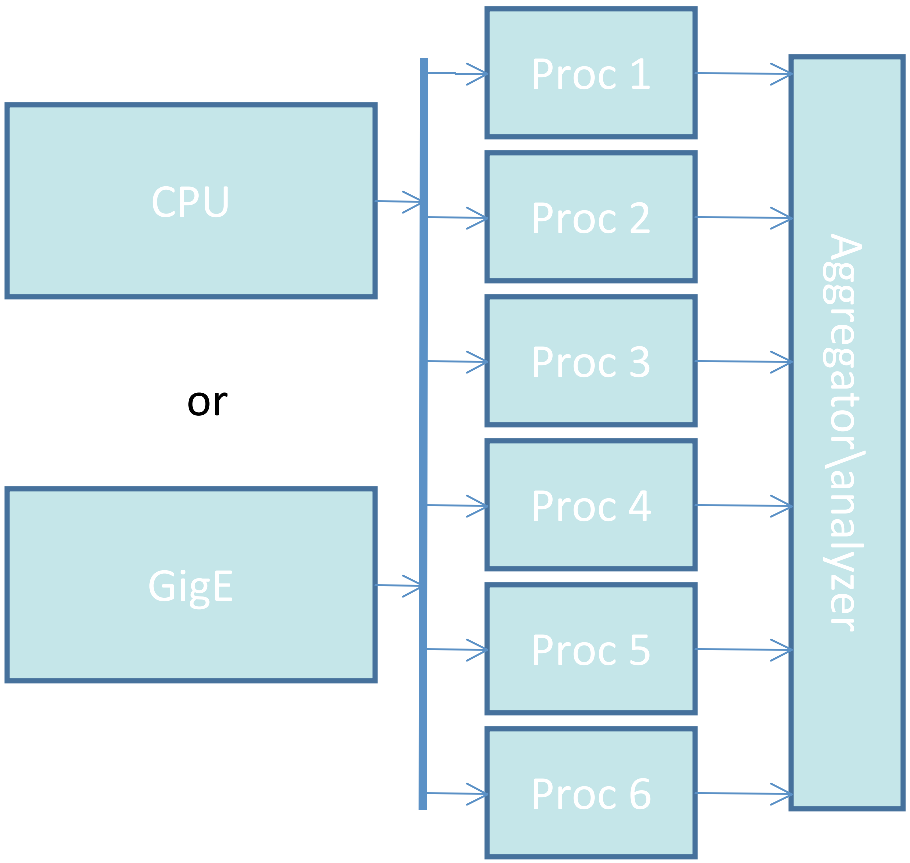
Tilera TFIDF mapping

- Goals: fit classifier dictionary in 64KB L2 cache of each tile; stream the data
- Approach
  - Use an array to hold a state machine: no tokenizing!
    - input character code is row index, current state is column index
    - array value contains next state and a key
    - when token terminator is read, the key associated with current state is incremented
    - Unknown token will hopefully fall off the paths and go into a waiting column.
  - Strength: linear in size of document, fits in memory
  - Weaknesses
    - increase false positive rate (255 strings per map)
    - Fairly complex array generator
    - Uses random number generation to generate the next index

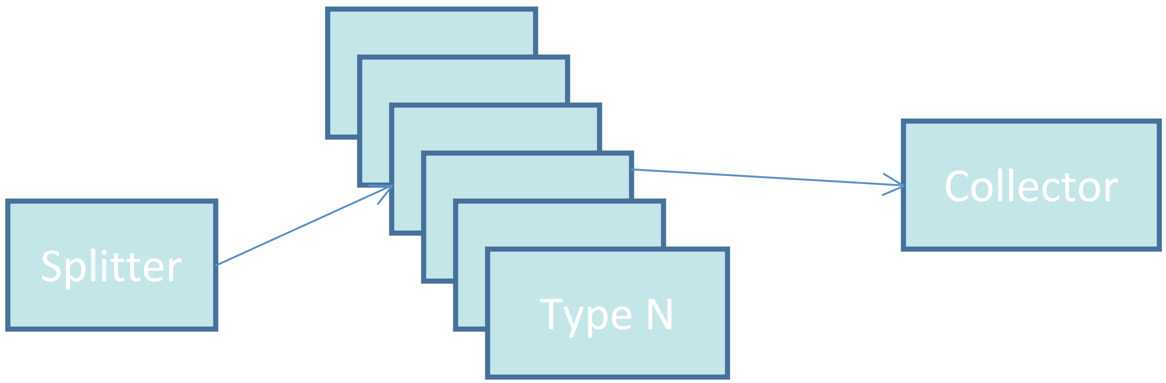
Layout

- Place a processing block for a single attack type in a single processor
- Use multiple processing blocks for parallel processing
- Each block processes all the different categories in parallel
- Run the data through in a streaming fashion
- Use as co-processor in conjunction with host CPU initially
- Stream packets off wire in production mode

Overall Layout



Processing layout



Example

Simple state machine with four terms

- Select
- Drop
- Odbc
- Statement

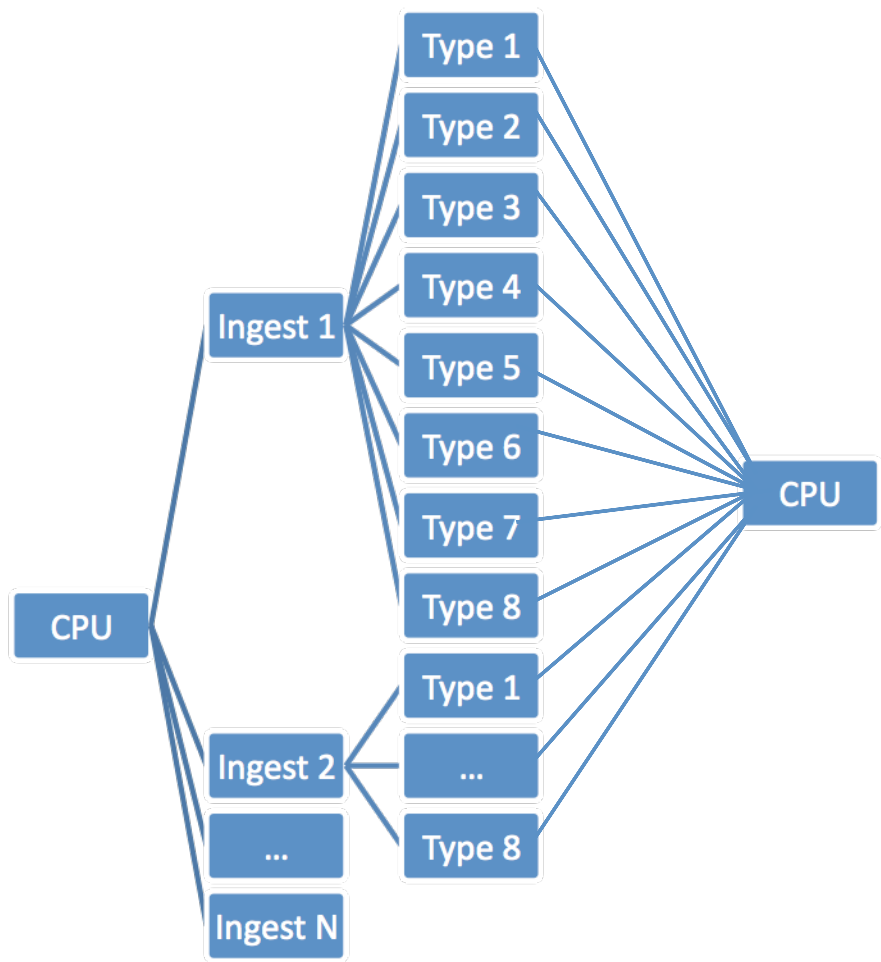
The rows representing letters contain the next column to examine

State Machine Structure

	0	1	2	3	4	5	6	7	8	9	10
A	0	0	4	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	9	0	0	0	0
C	0	0	0	0	0	0	0	0	6	1*3	0
D	0	2	0	6	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	3	0	8	5
F	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	9	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	10	0	0
N	0	0	0	0	0	9	0	0	0	0	0
O	0	3	0	0	10	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	1*2
Q	0	0	0	0	0	0	0	0	0	0	0
R	0	0	4	0	0	0	0	0	0	0	0
S	0	7	0	0	0	0	0	0	0	0	0
T	0	0	0	0	9	0	1*1	2	0	1*4	0
U	0	0	0	0	0	0	0	0	0	0	0
V	0	0	0	0	0	0	0	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	0	0	0	0	0	0	0
Z	0	0	0	0	0	0	0	0	0	0	0
space	1	1	1	1	1	1	1	1	1	1	1

Tilera Implementation

- Packets are transmitted from the CPU to the Tilera through the PCI bus using the zero copy transfer mechanisms.
- The CPU process is multithreaded on both transmit and receive.
- The Tilera ingest blocks receive the data from the CPU then transmit the data using broadcast messages to the individual processing blocks.
- Each processing block has a dedicated tile



Processing Blocks

- Blocks loop through the characters in the packet
- The tokens are counted, and at the end of the packet the score is computed for each type according to the formula.
  - The scores computation is fast due the fact that most of the matching tokens have 0 matches, so there are a lot of zeros which is fast even in a core without hardware floating point.

Tilera Implementation Performance

# Attack Types, Blocks	1	2	4	8
1	6.42	6.42	6.42	
2	12.79	12.79	12.77	12.7
3	19.09	19.09	19.09	18.99
4	25.34	25.34	25.34	25.22
5	31.58	31.58	31.58	31.39
6	37.7	37.7	37.58	**37.4**

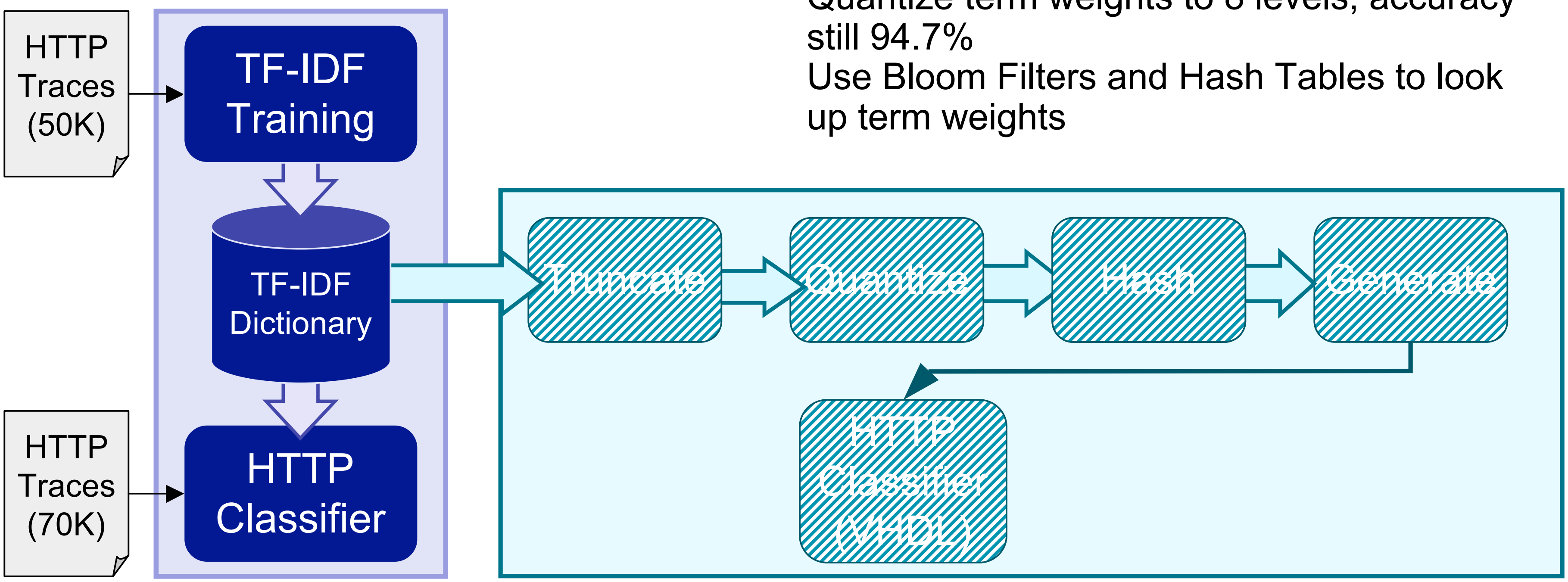
MB/s

- can trade off between number of concurrent processing units and number of attack types; best result is 73.55MB/s for 2 attack types
- 37X original implementation on single 20W chip

TFIDf on FPGA

- Simplify formula: classifier just gives attack indicator, not attack type
- Truncate term vector: 1948 terms

Quantize term weights to 8 levels; accuracy still 94.7%
Use Bloom Filters and Hash Tables to look up term weights



Craig Ulmer, Sandia CA

TFIDF on FPGA

- Simplify formula: classifier just gives attack indicator, not attack type
- Truncate term vector: 1948 terms

Lookup IDF for term in dictionary

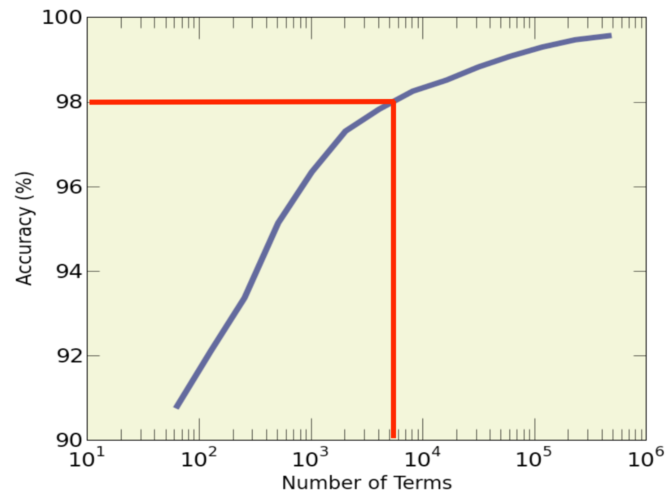
Count each term in input

Lookup weight for term in dictionary

$$score\{classifier\} = \frac{\sum_{t \in q \cap R} \cancel{\frac{count[t]}{\#input\ terms}} \cdot idf[t] \cdot ClassLabelTfidf[classifier][t]}{\cancel{\sqrt{\sum\left(\frac{count[t]}{\#input\ terms} \cdot idf[t]\right)^2}} \cdot DocMagnitude[classifier]}$$

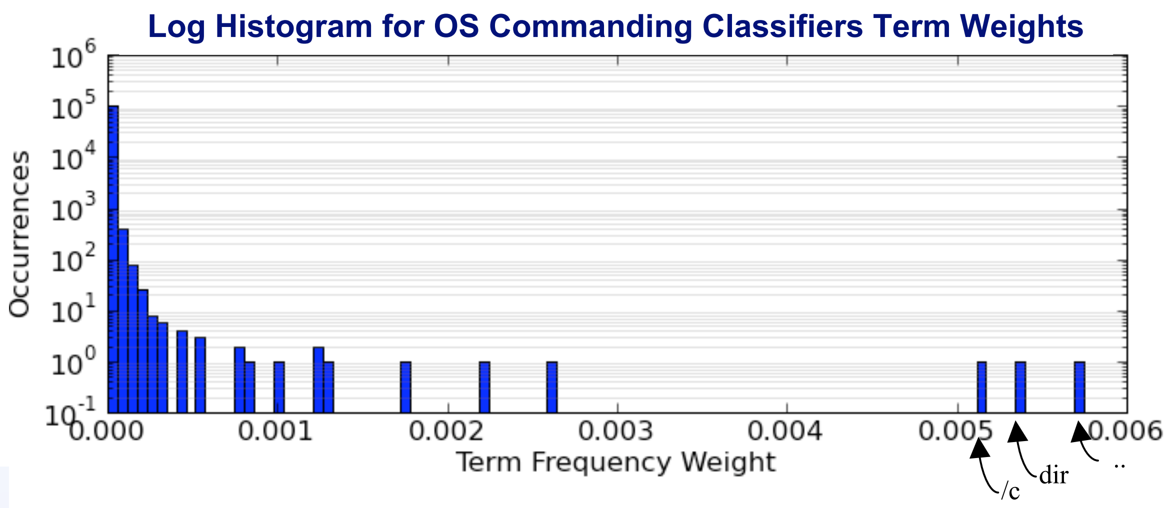
Scale based on TF-IDFs found by ALL classifiers

Adjust based on weight of classifier



Dictionary Observations

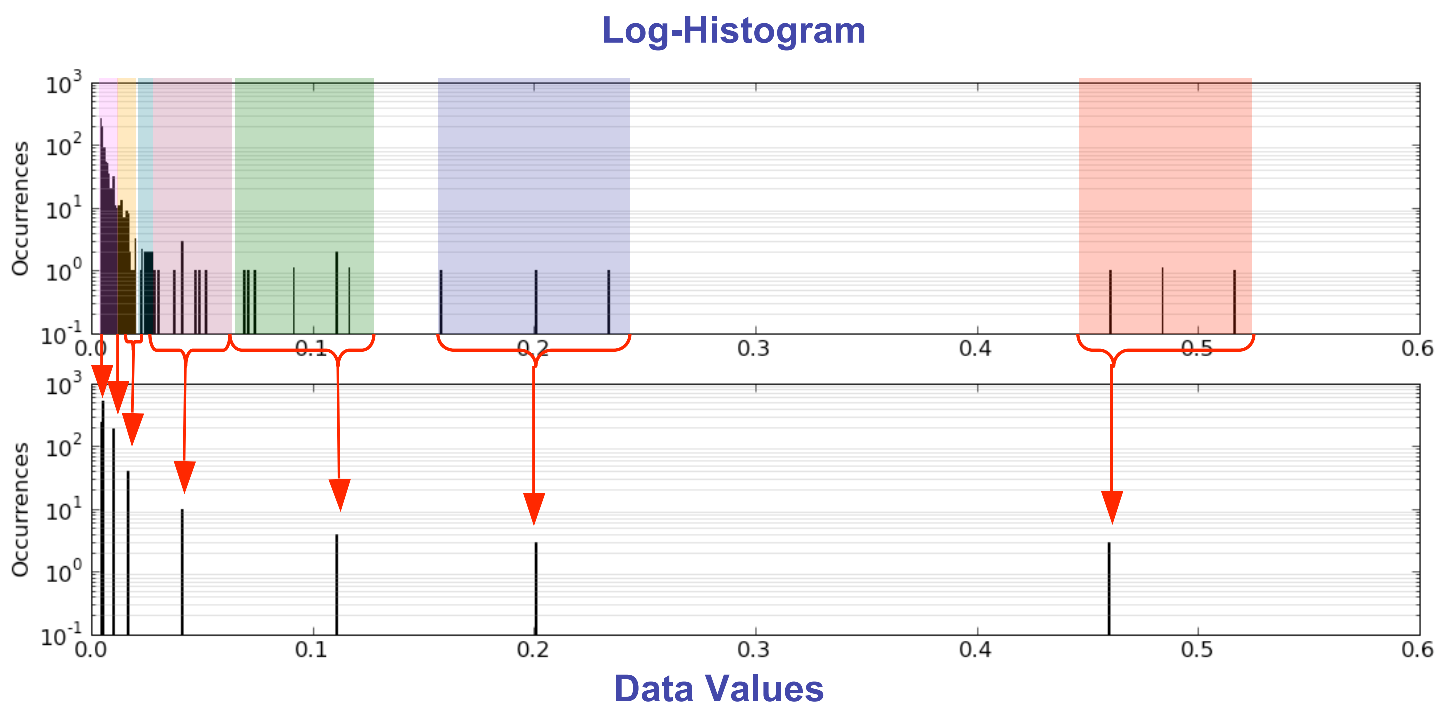
- Many terms in the dictionary
  - 1.8M terms (46MB text, 128MB data)
  - Many terms are junk ("rv:0.7.8"), but they also get very low weight
- Data values are not very diverse
  - Total unique values is < 2% of population
  - Eg: OS Classifier has 102K terms, but only *415* unique weights



Quantize Dictionary Term Weights

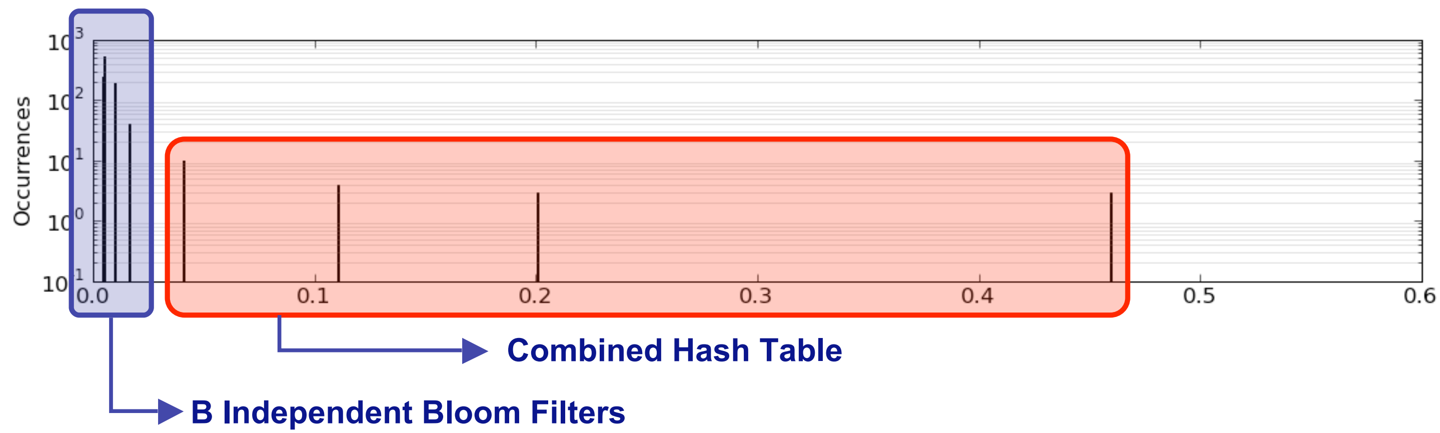
- How accurate do data values in dictionary need to be?
- Does IDF(“ODBC”) = 0.500001 give more accurate results than..
  - 0.500002?   0.488886?   0.03?
- Experiment:
  - Reduce unique data values in dictionary, measure accuracy impact

Re-Quantizing Data



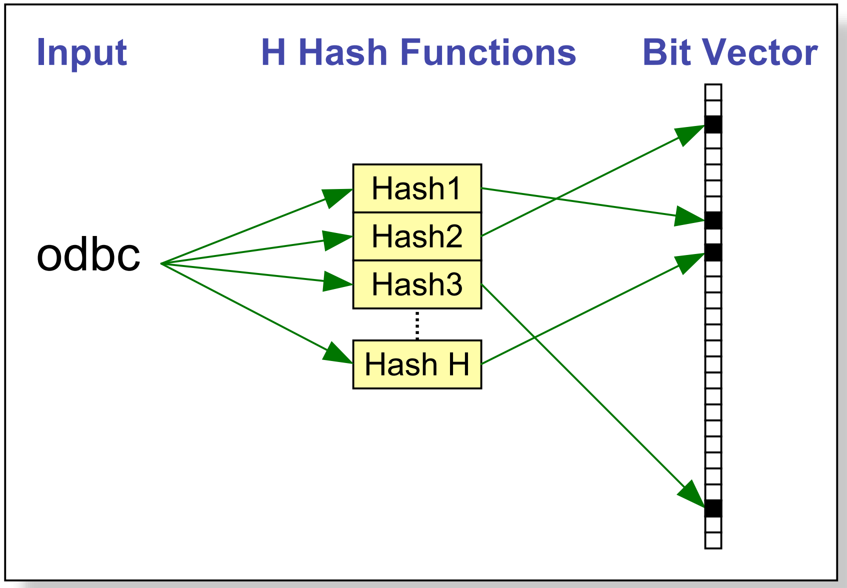
Hashing Tricks

- Small sets: combine into a single hash table
  - Brute-force packing sufficient for small tables
- Large sets: Array of Bloom filters
  - Bloom filters: space-efficient way to determine set membership
  - No false negatives, but can have false positives

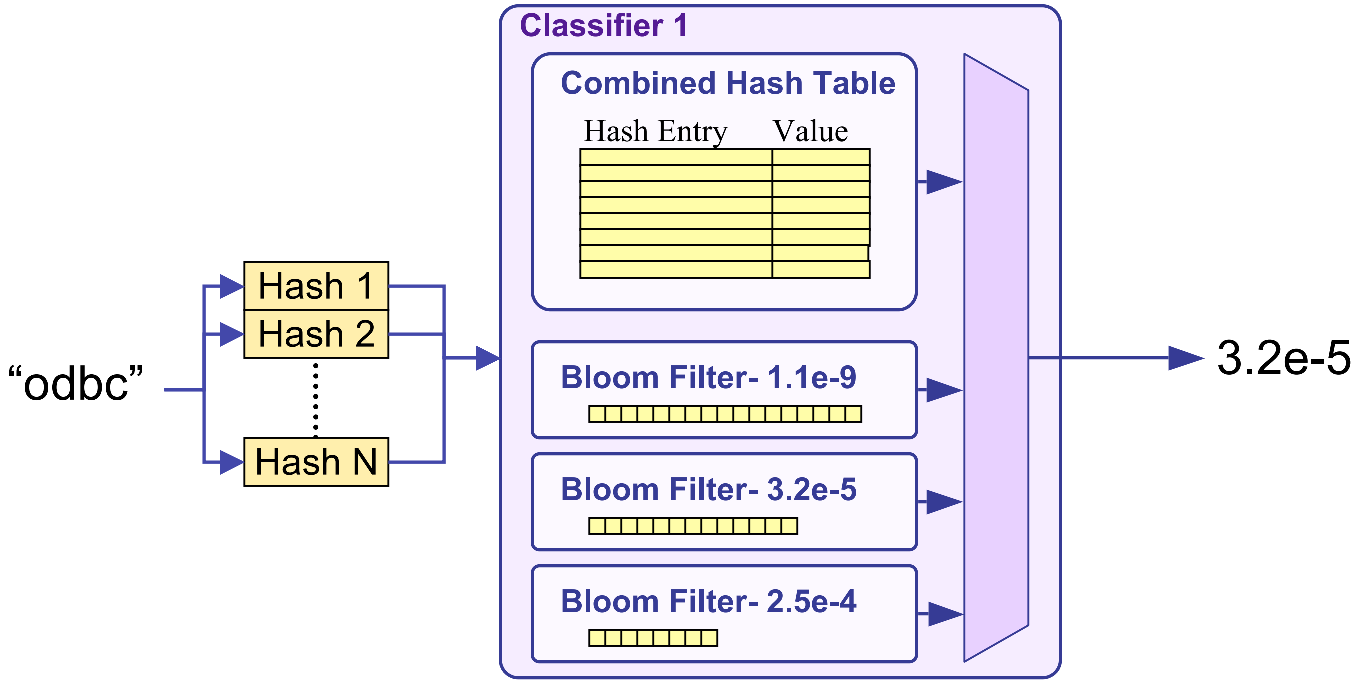


Bloom Filters

- Bloom filters: space-efficient way to test set membership
  - Given: list of set members (odbc, drop, table, ...)
  - Determine if an input belongs in set or not
  - Employ bit vector and H different hash functions



Hashing Replaces Dictionary



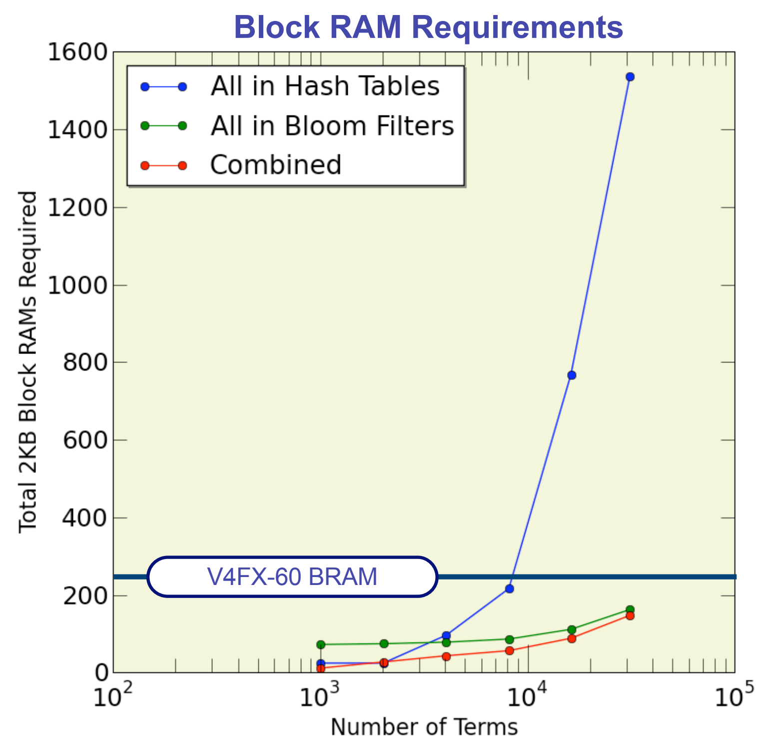
For 2KB Memory Block:

256 Hash table entries

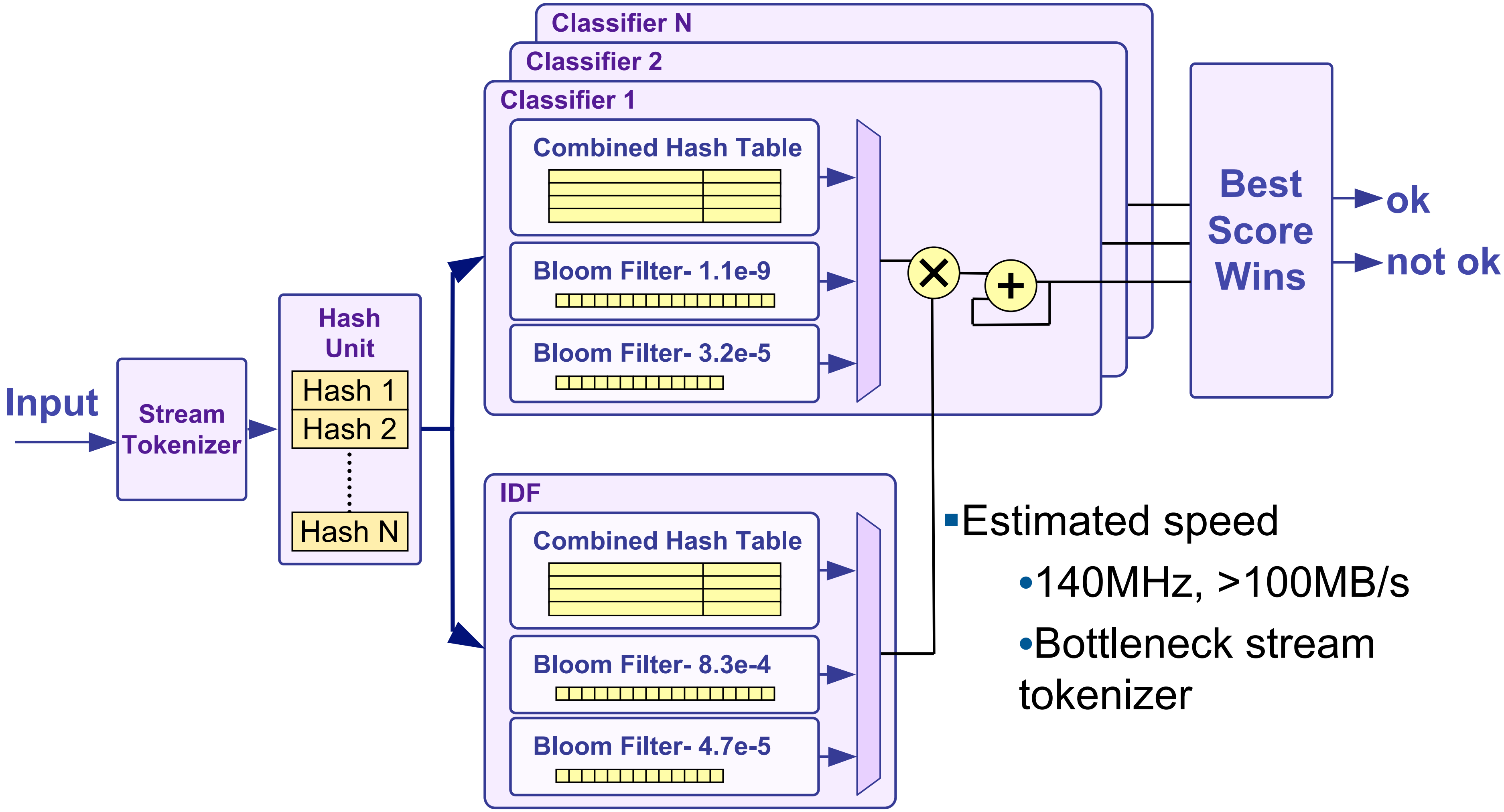
~1K Bloom Filter members

Generating Hardware

- Data set characteristics drive hardware design
- Use top k terms of term dictionary
- Truncate/quantize based on actual term frequency weights
- weight lookup method chosen based on number of terms at that weight
- Implemented flexible hardware design
  - Perl script converts data to parameters
  - parameters can generate C program or VHDL package
- Piecewise testing
  - Full design in simulation software
  - Testing on Xilinx ML555 Virtex5 board: read Ethernet packets, tokenize, stream into TF/IDF block



Hardware Data Flow



- Estimated speed
  - 140MHz, >100MB/s
  - Bottleneck stream tokenizer

Summary

- Data intensive problems require data-centric architectures and programming environments
- Study demonstrated data parallel and streaming approaches to a TFIDF web traffic classifier
- Hadoop: suitable for forensic analysis
  - < 1MB/s, 120W, 1 month
- Hardware-accelerated streaming approaches can take the data off the wire (or host)
  - compromise on accuracy for speed (94% accuracy instead of 95%)
  - select and customize data structures to fit available on-chip memory
  - Tilera: 37MB/s for 8 attack types, 73MB/s for 2 attack types, 20W, 3 months
  - FPGA: 140MB/s, 20-ish W, 6 months